SIERRA-PHP FRAMEWORK

by Jason Read

There is certainly no shortage of PHP frameworks these days. If you don't mind varying degrees of arm twisting, these frameworks can help you to create better software in less time. In this article, I'll provide an example of this by creating a blogging application in about 60 lines of PHP code using the sierra-php framework.

Like many frameworks, sierra-php started as a small personal library of reusable classes intended to reduce code redundancy. That was circa 2002 when PHP 4.2 was first released and the PHP framework proliferation had not yet begun. Today, with all of the great PHP frameworks readily available, I'll be the first to admit that creating a new framework is probably not the best use of one's time.

My intent in writing this article is to demonstrate some of the more useful and distinguishable features of sierra-php. In doing so, I'll try to be honest and objective and to keep the clichés and "*my framework solves world hunger*" fluff to a minimum.

Before we get started, I should mention a few hard and soft requirements of sierra-php. First of all, sierra-php can only be used on **nix* operating systems such as Linux, BSD or OS X. It cannot be used on Windows. Second, in terms of the convention versus

REQUIREMENTS

PHP: *4.3.4+*

Other Software:

- Unix-like (e.g. Linux, BSD, OS X)
- RDBMS (MySQL, PostgreSQL or SQLite)
- XCache, APC, or other PHP opcode cacher

Related URLs:

- Project Website: http://sierra-php.org
- Blog Demo: http://blog.sierra-php.org
- Download: http://sierra-php.org/download
- API Docs: http://api.sierra-php.org

configuration camps, sierra-php is on the side of the latter. It uses DTD-based XML for the configurable parts of your software. If the mere sight of XML gives you a queasy, uncomfortable feeling inside, you should probably stop reading now. Finally, sierra-php must be hosted in an environment where you are provided with shell access (e.g. SSH, although root permissions are not necessary). You will need this access in order to extract the sierra-php source, create an application, and use the *sierra-php console* for debugging.

sierra-php in a Nutshell

sierra-php utilizes a few useful design patterns to provide a foundation for creating PHP software. Most significant among these are *Model-View-Controller (MVC)*, *Data Access Object (DAO)* and *Value Object (VO)*, where the latter two are automatically generated from sierra-php's *Object-relational mapping (ORM)* XML markup. sierra-php also borrows some principles from *Aspect-Oriented Programming (AOP)* allowing some cross-cutting concerns, such as authentication, to be removed entirely from your code.

I think that is enough acronyms and marketing fluff...so let's get started with the blogging app and see how it actually works.

sierra-php Blog Requirements

Usually before you start writing any software, you receive some form of requirements documentation. For the blogging application, these are the requirements:

- **Standard blog functionality**: manage and add comments to posts
- **Upload picture**: a post may have an optional picture header
- **Authentication**: only authenticated users may manage posts
- **L10n**: user interface should support both English and Spanish
- **Web services**: in additional to the browser interface, blog functionality should also be available via *REST* and *SOAP* Web services
- **PDF print**: posts should be viewable in PDF format for printing

Setting Up sierra-php

First, you'll need to get a copy of sierra-php. The easiest way to do this is to download and extract the latest source archive from the sierra-php Google Code website (see Related Links). You may also checkout sierra-php from Subversion or install the RPM. For the purpose of this article, I'll be using the sierra-php source archive installed on an Ubuntu shared hosting account. After extracting the source archive, you will be left with a single directory named *sierra*. Now, you'll just need to run the sierra-php setup script **sierra/bin/sra-quick-install.php** (this requires the *php* executable to be in your path). This script will perform some necessary setup tasks prior to using the framework. You can see a summary of my shell session in Listing 1:

Setting Up the Blog Application

sierra-php organizes software into applications. By default, these applications are located in **sierra/app** (e.g. **sierra/app/blog**). Each application directory must contain a few sub-directories including:

- **etc**: contains application configuration files
- **etc/l10n**: contains language specific properties files
- **lib**: contains PHP classes and source files that are not web visible
- **www/html**: contains web visible PHP scripts and static files
- **www/tpl**: contains *Smarty* templates used for rendering views

The names of these directories as well as the location of the base application directory are all configurable, but for this article, we'll just use the defaults. sierra-php includes an installer script that can be used to quickly setup a new application: **sierra/bin/sra-installer.php**. If you've run **sra-quick-install.php** and if the installer can locate the {{php}} binary, this script will be executable so that it can be run directly. Once started, it will show a CLI menu providing the installer options. Select the

LISTING 1

```
shell> wget http://sierra-php.googlecode.com/files/sierra-php-1_1_3.tgz
shell> tar zxf sierra-php-1_1_3.tgz
shell> rm -f sierra-php-1_1_3.tgz
shell> php sierra/bin/sra-quick-install.php
```

option *[2] Configure application* followed by *[1] Install new application* (twice). The installer will then prompt you with a few questions about your application. For most of these, you can just use the default responses. For this article, I named my application *blog*. Finally, since I'm setting this up on a shared hosting server, I need to link the application's web visible directory **sierra/app/blog/www/html** to my **public_html** directory. Here is a summary of my shell session:

```
shell> sierra/bin/sra-installer.php
shell> rmdir sierra/app/blog/www/html
shell> ln -s ~/public_html sierra/app/blog/www/html
```

Configuring the Blog Database

sierra-php provides a database abstraction layer with support for MySQL, PostgreSQL, SQLite, and SQL Server. Because sierra-php is still PHP 4 compatible, it does not utilize *PDO*. In order to utilize a database in a sierra-php application, the connection parameters for that database must be declared in the application configuration file **sierra/app/blog/etc/app-config.xml**. For this article, I'll be using a MySQL database named *blog*, username *sierraphp*, and password *passw0rd*. Listing 2 shows the updated application configuration file.

LISTING 2

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE app-config PUBLIC "-//SIERRA//DTD APP CONFIG//EN"
  "http://sierra-php.googlecode.com/svn/trunk/etc/app-config.dtd">
<app-config>
  <db key="blog"
      user="sierraphp"
      password="passw0rd" />
</app-config>
```

LISTING 3

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE entity-model PUBLIC "-//SIERRA//DTD ENTITY MODEL//EN"
  "http://sierra-php.googlecode.com/svn/trunk/lib/model/entity-model.dtd">
<entity-model>
  <entity key="BlogPost" primary-key="postId">
    <attribute key="postId" sequence="1" />
    <attribute key="author" max-length="16" />
    <attribute key="comments" cardinality="0..*"
               type="BlogPostComment" />
    <attribute key="post" depends="required" />
    <attribute key="posted" read-only="1" type="time" />
    <attribute key="title" depends="required" max-length="64" />
  </entity>

  <entity key="BlogPostComment" primary-key="commentId">
    <attribute key="commentId" sequence="1" />
    <attribute key="comment" depends="required" />
    <attribute key="email" depends="email required" max-length="64" />
    <attribute key="name" depends="required" max-length="16" />
    <attribute key="posted" read-only="1" type="time" />
  </entity>

  <msg key="email" resource="error.email" />
  <msg key="maxLength" resource="error.maxLength" />
  <msg key="required" resource="error.required" />
</entity-model>
```

sierra-php Models

sierra-php provides an XML markup for describing an application's model. This model describes the persistable classes, class associations, data validation, and views used by the application. Once the model is complete, or whenever it is changed, sierra-php generates the *Data Access Object (DAO)* vand *Value Object (VO)* PHP classes, and optionally creates or updates the application's database schema. The sierra-php model markup differs from that of ORMs in that classes are described according to how they are used in the software (i.e. object-oriented design), not how they are stored in the database (i.e. relation design), which are often very different.

> sierra-php utilizes a few useful design patterns to provide a foundation for creating PHP software.

Once the DAO and VO classes have been generated by sierra-php for an application, they may be used to create, retrieve, update and delete (CRUD) those instances of the class in your software without writing database access or SQL code.

Defining the Blog Model

We'll define two classes initially in the blog model: **BlogPost** and **BlogPostComment**. To do so, we'll use the *entity* element. Each *entity* element contains one or more *attribute* sub-elements which define the properties of that class. Listing 3 shows the resulting model. A few things to point out are:

- Entities must have primary keys in order to be uniquely referenced. In the blog model, those identifiers are *postId* and *commentId* which are both sequences which signify that the database will handle incrementing this value (e.g. **AUTO_INCREMENT** in MySQL)
- Because a blog post can have zero-to-many comments associated to it, the *comments* attribute utilizes the special property *cardinality*

to designate that association. This will translate into the addition of a foreign key column in the table for blog post comments when the schema is generated

- The *depends* property designates validation rules associated with that attribute. sierra-php provides a handful of common validation rules and developers may also implement their own custom rules. *required* and *email* are used in this model to designate attributes that are mandatory and those that must be a properly formatted email address. These validation rules can be evaluated by invoking the *validate* method provided by the VO. The DAO will invoke this method whenever an attempt is made to insert or update an instance of the entity. When validation failures occur, the strings designated by the *msg* elements at the bottom of the model will be set to the **validationErrors** property of the VO

- The attribute *type* designates the data type. The default type is a *string*. Other supported types include *blob*, *boolean*, *date*, *float*, *int* and *time*. *max-length* is a shorter way of defining the maximum length validation rule for *string* type attributes

Supporting Localization

A big part of software localization (L10n) is multiple language support. One of the HTTP request headers sent by browsers (**$_SERVER['HTTP_ACCEPT_LANGUAGE']**) is an ordered list of locale preferences (language + country) designated by the user. sierra-php utilizes that header in combination with Java style properties or string files (e.g. *app.properties*, *app_es.propeties*, *app_es_mx.properties*), to select and use the language strings that match the highest possible user locale preference. These strings can then be referenced using the **SRA_ResourceBundle** class as well as in Smarty templates using the automatically-added reference to the application's main properties file **$resources** (e.g. **$resources->getString('text.hello')**). L10n support in sierra-php extends to the model as well. Each generated VO class contains two methods, **get[AttributeName]Label** and **get[AttributeName]HelpContent** allowing you to retrieve localized property labels and descriptions. To accomplish this, we need to add strings for those properties where the string key is formatted *[entity name].[attribute name]* for the label and *[entity name].[attribute name]-help* for the description. Listing 4 shows the English strings for our first two classes.

Creating Model Views

Now that we have defined two classes and their associated properties, we can now define their views. A view is essentially a way of displaying an object

FIGURE 1

LISTING 4

```
BlogPost=Post
BlogPost.postId=Post ID
BlogPost.postId-help=The unique identifier for this blog post
BlogPost.author=Author
BlogPost.author-help=The name of the blog post author
BlogPost.comments=Comments
BlogPost.comments-help=The comments submitted for this blog post
BlogPost.post=Post
BlogPost.post-help=The blog post content
BlogPost.posted=Post Time
BlogPost.posted-help=The date and time when this post was created
BlogPost.title=Title
BlogPost.title-help=The blog post title

BlogPostComment=Comment
BlogPostComment.commentId=Comment ID
BlogPostComment.commentId-help=The unique identifier for this comment
BlogPostComment.comment=Comment
BlogPostComment.comment-help=The comment being made
BlogPostComment.email=Email Address
BlogPostComment.email-help=The email address of the comment author
BlogPostComment.name=Name
BlogPostComment.name-help=The name of the comment author
BlogPostComment.posted=Submit Time
BlogPostComment.posted-help=Date and time when comment was submitted

error.email={$attr} is a not properly formatted email address
error.maxFileSize=The maximum size for files for {$attr} is 100 KB
error.maxLength=The max length of {$attr} is {$maxLength} characters
error.mimeType=Please upload only file of type: {$mimeTypes}
error.required={$attr} is required
```

LISTING 5

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE entity-model PUBLIC "-//SIERRA//DTD ENTITY MODEL//EN"
  "http://sierra-php.googlecode.com/svn/trunk/lib/model/entity-model.dtd">
<entity-model>
  <entity key="BlogPost" primary-key="postId">
    <attribute key="postId" sequence="1">
      <view key="input" extends="input-hidden" />
    </attribute>
    <attribute key="author" max-length="16" />
    <attribute key="comments" cardinality="0..*" type="BlogPostComment" />
    <attribute key="post" depends="required">
      <view key="input" extends="input-textarea" />
    </attribute>
    <attribute key="posted" read-only="1" type="time" />
    <attribute key="title" depends="required" max-length="64">
      <view key="input" extends="input-text" />
    </attribute>

    <view key="form" template="views/blog-post-form.tpl" />
    <view key="html" template="views/blog-post.tpl" />
  </entity>

  <entity key="BlogPostComment" primary-key="commentId">
    <attribute key="commentId" sequence="1">
      <view key="input" extends="input-hidden" />
    </attribute>
    <attribute key="comment" depends="required">
      <view key="input" extends="input-textarea" />
    </attribute>
    <attribute key="email" depends="email required" max-length="64">
      <view key="input" extends="input-text" />
    </attribute>
    <attribute key="name" depends="required" max-length="16">
      <view key="input" extends="input-text" />
    </attribute>
    <attribute key="posted" read-only="1" type="time" />

    <view key="form" template="views/blog-post-comment-form.tpl" />
    <view key="html" template="views/blog-post-comment.tpl" />
  </entity>

  <global-views>
    <view key="image" template="model/sra-attr.tpl">
      <param id="value" type="tpl" value="model/sra-html-img.tpl" />
    </view>
    <view key="input-file" template="model/sra-attr.tpl">
      <param id="value" type="tpl" value="model/sra-form-file.tpl" />
      <param id="showResetLink" value="text.reset" />
    </view>
    <view key="input-hidden" template="model/sra-attr.tpl">
      <param id="value" type="tpl" value="model/sra-form-input.tpl" />
      <param id="type" type="input-attrs" value="hidden" />
    </view>
    <view key="input-text" template="model/sra-attr.tpl">
      <param id="value" type="tpl" value="model/sra-form-input.tpl" />
    </view>
    <view key="input-textarea" template="model/sra-attr.tpl">
      <param id="useTextArea" value="1" />
      <param id="value" type="tpl" value="model/sra-form-input.tpl" />
    </view>
  </global-views>

  <msg key="email" resource="error.email" />
  <msg key="maxFileSize" resource="error.maxFileSize" />
  <msg key="maxLength" resource="error.maxLength" />
  <msg key="mimeType" resource="error.mimeType" />
  <msg key="required" resource="error.required" />
</entity-model>
```

FIGURE 2

or property. Views are added to the model using the *view* sub-element within either *entity* or *attribute*. The *view* sub-element designates a name for that view along with the path to a Smarty template that will be used to render it. sierra-php provides some re-useable templates for rendering common views such as *input* HTML fields. The sierra-php model XML markup also allows for extensible views so developers do not need to repetitively define similar views. Listing 5 shows the new blog model with views added. Listing 6 shows the Smarty template for BlogPost's *form* view, and Figure 1 shows that view in a browser window. Listing 7 shows the Smarty template for BlogPost's *html* view, and Figure 2 shows that view in a browser window. Note, that the view templates aren't meant to be entire HTML documents, but rather fragments of HTML that constitute that particular view.

> One of the most powerful features of the sierra-php model is the ability to add both REST and SOAP Web services to your application declaratively.

Adding Blog Post Pictures

You may have noticed a reference to *picture* in those listings and figures. The idea for this is to allow a blog post to include a picture displayed directly below the title. If you have worked with file uploads and storing files in a database, you will know what a huge hassle it can be to validate, store and re-render those files. sierra-php makes this a little easier by providing file-type attributes, file validation, and file rendering. To support this in the blog model, we need to add a new property to **BlogPost** named *picture*:

```
<attribute key="picture"
    depends="maxFileSize mimeType" is-file="1">
  <var key="mimeTypes" value="image.*" />
  <var key="maxFileSize" value="102400" />
  <view key="input" extends="input-file" />
  <view key="output" extends="image" />
</attribute>
```

The result of adding this property is a new column in the blog post table which uses MySQL's *blob* data type. Additionally, when a picture is uploaded for a blog post, it will be validated such that it must be an image mime-type and less than 100 KB in size. When the *output* view for *picture* is rendered, it will generate an HTML *img* element where *src* is a link to the sierra-php rendering script which will display the image including pulling it from the database and providing the browser with the correct HTTP response headers including *Content-type* and *Content-length*.

Adding Authentication

Since we don't want just anyone posting on our blog, we'll need to add authentication restricting access to the screens that allow users to create, update or delete posts. Since authentication is such a common application requirement, sierra-php provides a means of adding it declaratively in an Aspect-Oriented manner using the application configuration file. To do so, you must first define an authenticator which is essentially a persistent user repository. sierra-php currently supports four such repositories: database, LDAP, operating system and POP3. An application may define any number of user repositories and authentication can occur against one, or multiple of those repositories in an inclusive or exclusive manner (i.e. user must authenticate against all or just one repository when multiple are specified). For our blog app, we'll just be using one database type repository.

Once an authenticator is defined in the application configuration file, the next step is to restrict access to the application's Web accessible PHP scripts. This is also done in the same configuration file using the *restrict-access* element, where each such element associates authenticator(s) with the scripts that should be protected. When a user attempts to access one of those protected scripts, sierra-php will utilize standard HTTP authentication headers to prompt and validate users. sierra-php also supports user logout without requiring the user to close their browser window (usually a shortcoming of HTTP authentication). Listing 8 shows the new application configuration file

LISTING 6

```
<label>{$entity->getTitleLabel()}</label>:
{$entity->renderAttribute('title', 'input')}
<p>{$entity->getTitleHelpContent()}</p>
<hr />

<label>{$entity->getPostLabel()}</label>:
{$entity->renderAttribute('post', 'input')}
<p>{$entity->getPostHelpContent()}</p>

<label>{$entity->getPictureLabel()}</label>:
{$entity->renderAttribute('picture', 'input')}
<p>{$entity->getPictureHelpContent()}</p>

<input name="author" type="hidden" value="{$user.name}" />
```

LISTING 7

```
<hr />
<h2>{$entity->getTitle()}</h2>
{if $entity->getPicture()}
  <p>{$entity->renderAttribute('picture', 'output')}</p>
{/if}
<p>
  <strong>
    {if $entity->getAuthor()}{$entity->getAuthor()} - {/if}
    {$entity->getPosted(0, 1)}
  </strong>
  <a href="...">{$resources->getString('text.edit')}</a> |
  <a href="#" onclick="...">{$resources->getString('text.delete')}</a>
  <br />
  {$entity->getPost()}
  <h3>{$entity->getCommentsLabel()}</h3>
  <p><a href="...">{$resources->getString('blog.submitComment')}</a></p>
{if $entity->getComments()}
{foreach from=$entity->getComments() item=comment}
  {$comment->render('html')}
{/foreach}
{else}
  {$resources->getString('blog.noComments')}
{/if}
</p>
```

LISTING 8

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE app-config PUBLIC "-//SIERRA//DTD APP CONFIG//EN"
  "http://sierra-php.googlecode.com/svn/trunk/etc/app-config.dtd">
<app-config error-log-file="blog.log">
  <authenticator key="blog"
                 logout-fwd-uri="/"
                 logout-get-var="logout"
                 resource="text.login"
                 sys-err-tpl="error.tpl"
                 tpl-var="user"
                 type="SRA_DbAuthenticator">
    <param id="db" value="blog" />
    <param id="table" value="blog_admin" />
    <param id="user-col" value="user_name" />
    <param id="pswd-col" value="password" />
    <param id="pswd-fun" value="md5" />
    <param id="name" type="attrs" value="name" />
  </authenticator>

  <db key="blog"
      host="db1"
      name="sierraphp_blog"
      user="sierraphp"
      password="sierraphp" />

  <restrict-access authenticators="blog"
          match="delete-comment.php|delete.php|edit.php|post.php" />

  <use-entity-model key="blog" path="blog-model.xml"/>
</app-config>
```

LISTING 9

```
<entity key="BlogAdmin" primary-key="adminId">
  <attribute key="adminId" sequence="1" />
  <attribute key="name" depends="required" max-length="16" />
  <attribute key="password" depends="required" min-length="4"
             max-length="255" set-function="md5" set-only="1" />
  <attribute key="userName" depends="required" max-length="16" />
</entity>
```

FIGURE 3

sierra-php Blog Web Services Overview

Web Services Gateway

The URI for this application's web service gateway is `http://blog.sierra-php.org/ws`

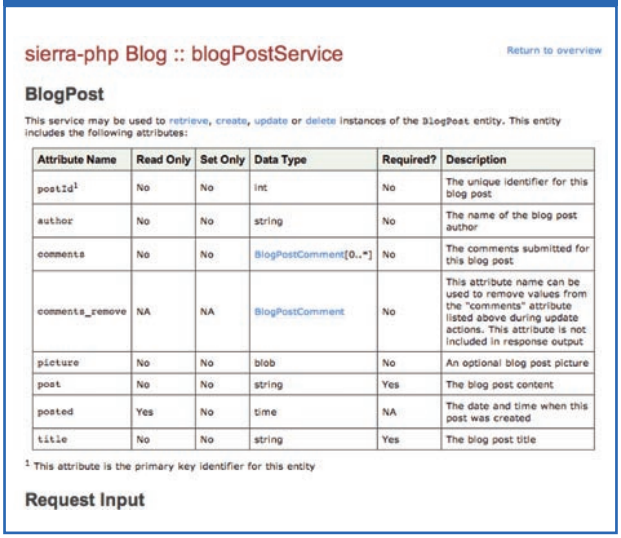
Web Services

The web services available for this application are:

Service Name	API Documentation	WSDL
blogPostService	View	Download
blogPostCommentService	View	Download

Download WSDL for all services

FIGURE 4



sierra-php Blog :: blogPostService

Return to overview

BlogPost

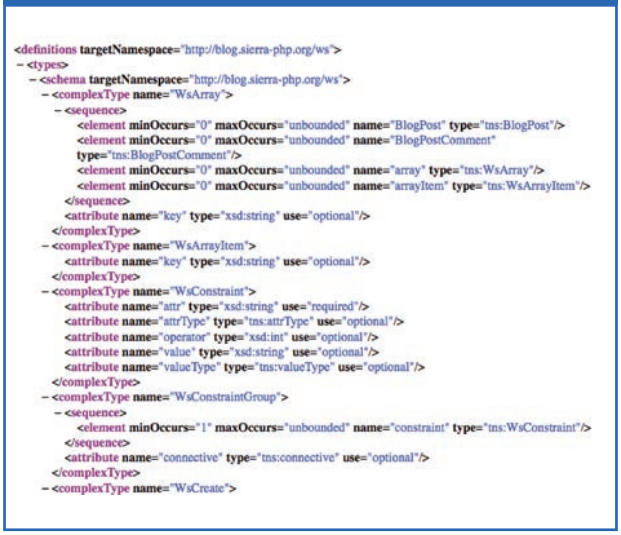
This service may be used to retrieve, create, update or delete instances of the BlogPost entity. This entity includes the following attributes:

Attribute Name	Read Only	Set Only	Data Type	Required?	Description
postId[1]	No	No	int	No	The unique identifier for this blog post
author	No	No	string	No	The name of the blog post author
comments	No	No	BlogPostComment[0..*]	No	The comments submitted for this blog post
comments_remove	NA	NA	BlogPostComment	No	This attribute name can be used to remove values from the "comments" attribute listed above during update actions. This attribute is not included in response output
picture	No	No	blob	No	An optional blog post picture
post	No	No	string	Yes	The blog post content
posted	Yes	No	time	NA	The date and time when this post was created
title	No	No	string	Yes	The blog post title

[1] This attribute is the primary key identifier for this entity

Request Input

FIGURE 5



```
<definitions targetNamespace="http://blog.sierra-php.org/ws">
- <types>
  - <schema targetNamespace="http://blog.sierra-php.org/ws">
    - <complexType name="WsArray">
      - <sequence>
          <element minOccurs="0" maxOccurs="unbounded" name="BlogPost" type="tns:BlogPost"/>
          <element minOccurs="0" maxOccurs="unbounded" name="BlogPostComment"
          type="tns:BlogPostComment"/>
          <element minOccurs="0" maxOccurs="unbounded" name="array" type="tns:WsArray"/>
          <element minOccurs="0" maxOccurs="unbounded" name="arrayItem" type="tns:WsArrayItem"/>
        </sequence>
        <attribute name="key" type="xsd:string" use="optional"/>
      </complexType>
    - <complexType name="WsArrayItem">
        <attribute name="key" type="xsd:string" use="optional"/>
      </complexType>
    - <complexType name="WsConstraint">
        <attribute name="attr" type="xsd:string" use="required"/>
        <attribute name="attrType" type="tns:attrType" use="optional"/>
        <attribute name="operator" type="xsd:int" use="optional"/>
        <attribute name="value" type="xsd:string" use="optional"/>
        <attribute name="valueType" type="tns:valueType" use="optional"/>
      </complexType>
    - <complexType name="WsConstraintGroup">
      - <sequence>
          <element minOccurs="1" maxOccurs="unbounded" name="constraint" type="tns:WsConstraint"/>
        </sequence>
        <attribute name="connective" type="tns:connective" use="optional"/>
      </complexType>
    - <complexType name="WsCreate">
```

with authentication added. The authenticator's *resource* property (**text.login**) references a string in the application's properties file that will be displayed in the browser's popup login dialog.

Now that we have defined the authenticator, we need to create the *blog_admin* table that sierra-php will try to authenticate against. To do so, we'll add a new class to the blog model. Listing 9 shows the XML we'll use to do so. Note the use of MySQL's `MD5()` function used by the *password* property, which means that passwords are not stored in the clear. The `MD5()` function is also referenced in the authenticator configuration. Note also that the authenticator configuration in Listing 8 defines a *tpl-var* property named **user**. This tells sierra-php to automatically set the user properties (username and password, as well as the additional property *name* from the *name* column in the *blog_admin* table) to an associative array which can be referenced in Smarty templates using the variable **user**. Listing 6 uses the user's name to automatically set a new post's *author* property using a hidden input field.

```
<input name="author" type="hidden"
   value="{$user.name}" />
```

Adding Web Services

Web services provide not only a means of exposing data and functionality to external systems, but also incorporating RIA functionality into your browser interface via AJAX and/or Flex. One of the most powerful features of the sierra-php model is the ability to add both REST and SOAP Web services to your application declaratively. To do so, add a *ws* element to an *entity* in the model designating the name of the Web service along with what CRUD functionality it should allow. For example, to add a Web service providing full CRUD access to **BlogPost**, the following XML would be added:

```
<ws key="blogPostService" create="1" delete="1"
   update="1" />
```

Once this Web service is defined, sierra-php will also publish both an HTML-based API as well as the WSDL for SOAP clients. Figures 3, 4 and 5 provide examples of these for the new *blogPostService*.

In addition to exposing Web services that provide CRUD functionality for your entities, the application model can also be used to define a few other types of Web services that can be automatically generated and managed by sierra-php. including:

Any PHP function: You may define a Web service that executes a PHP function. Supports input parameters as well as multiple output data types including

arrays, hashes, dates, and scalar values. For example, the following XML model code exposes a Web service that invokes the static method **wsGetUserName** in the class **utils/DS_Users.php**.

```
<ws-global key="getUserNameService"
   identifier="utils/DS_Users::wsGetUserName" />
```

If you use PHPDoc formatted code comments, sierra-php will use those comments when constructing the automatically generated HTML API documentation for this Web service. This type of Web service requires you to implement the code in the **wsGetUserName** function.

Contents of properties files: Allows you to expose the contents of a properties file. If multiple versions of the properties file exist for different languages, sierra-php will automatically select the best matching properties file based on the browser's locale preferences. For example, the following XML model code exposes the contents of **etc/l10n/web-strings.properties** via a Web service:

```
<ws-global key="getAppStringsService"
   identifier="web-strings" type="rb" />
```

This type of Web service does not require you to write any code.

A database SQL query: Allows you to expose the results of a database SQL query. For example, the following XML model code exposes a Web service that retrieves the *id* and *user_name* values from the *users*:

```
<ws-global key="userLookup"
   identifier="SELECT id, user_name FROM users
   WHERE name like '#name#%'" type="sql">
```

It also defines a query parameter, *name* which will be used to filter the query results. SQL type Web service calls can also utilize pagination parameters to limit result sets (i.e. query *limit* and *offset* parameters). This type of Web service does not require you to write any code.

The DTD documentation for the *ws-global* element in **entity-model.dtd** provides more thorough documentation related to how to expose these additional types of Web services.

LISTING 10

```
<?xml version="1.0" encoding="UTF-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="main" margin="0in"
                           page-height="11in" page-width="8.5in">
      <fo:region-body margin="0.5in" margin-left="0in" />
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="main">
    <fo:flow flow-name="xsl-region-body" color="#000000"
             font-family="Helvetica" font-size="11pt">
      <fo:block font-size="16pt" font-weight="bold"
                margin-top="0.1in"
                text-align="center">{$entity->getTitle()|escape}</fo:block>
{if $entity->getPicture()}
{assign var=picture value=$entity->getPicture()}
      <fo:block text-align="center">
        <fo:external-graphic src="url('{$Controller->getServerUri()}
{$picture->getUri()|escape}')" />
      </fo:block>
{/if}
      <fo:block-container margin-left="0.4in" margin-top="0.2in">
        <fo:block font-weight="bold">
          {if $entity->getAuthor()}{$entity->getAuthor()|escape} - {/if}
          {$entity->getPosted(0, 1)}
        </fo:block>
        <fo:block>{$entity->getPost()|escape}</fo:block>
{if $entity->getComments()}
        <fo:block font-weight="bold">{$entity->getCommentsLabel()|escape}</
fo:block>
{foreach from=$entity->getComments() item=comment}
        <fo:block>
          {$comment->getName()|escape} ({$comment->getPosted(0, 1)}):
          {$comment->getComment()|escape}
        </fo:block>
{/foreach}
{else}
        <fo:block>{$resources->getString('blog.noComments')|escape}</
fo:block>
{/if}
      </fo:block-container>

    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

LISTING 11

```
include_once('../../../../lib/core/SRA_Controller.php');
SRA_Controller::init('sierraphp-blog');
$dao =& SRA_DaoFactory::getDao('BlogPost');
$post = BlogPost::newInstanceFromForm();

// if the form has been submitted, validate the blog post, insert and
redirect
// the user to the homepage
if (count($_POST) && $post->validate() && $dao->insert($post)) {
  header('Location: /');
}

$tpl =& SRA_Controller::getAppTemplate();
$tpl->assignByRef('post', $post);
$tpl->display('post.tpl');
```

LISTING 12

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="{$Locale->getId()}"
xml:lang="{$Locale->getId()}">
  <head>
    <title>{$resources->getString('blog.post.title')}</title>
  </head>
  <body>
    <h1>{$resources->getString('blog.post.title')}</h1>
{if $post->validateErrors}
    <font style="color:red">
    <ul>
{foreach from=$post->validateErrors item=error}
      <li>{$error}</li>
{/foreach}
    </ul>
    </font>
{/if}
    <form action="" method="post" enctype="multipart/form-data">
{$post->render('form')}
      <input type="submit" />
    </form>
  </body>
</html>
```

Adding a PDF BlogPost View

In addition to defining HTML views, sierra-php allows developers to pipe Smarty template-generated text into CLI executable programs to produce other types of view artifacts. We'll use this feature to add a new PDF view to **BlogPost** which will provide users with a print-friendly view of the post. To do so, we'll use the view's Smarty template to generate an XSL-FO document and Apache FOP to transform that document into a PDF view of the blog post. If you are not familiar with XSL-FO, it is a W3C standard like XHTML, but designed with the intent of being transformed into a printable document, usually PDF, and Apache FOP is a free open source implementation of that standard.

The first step to adding this view (assuming you have already installed Apache FOP) is to define a *view-processor* in the blog model for Apache FOP. This is basically the CLI path and arguments that should be used when piping in the XSL-FO generated by the Smarty template into Apache FOP:

```
<view-processor key="fop"
    args="{$outputFile} {$randomFile1}"
    output-file-path="{$randomFile1}"
    path="/usr/local/bin/fop" />
```

Next, we define a new view for **BlogPost** which utilizes the FOP view processor we just defined, as well as specifies a custom mime-type which will result in a *Content-type: application/pdf* response header when the view is rendered.

```
<view key="pdf"
    mime-type="application/pdf"
    view-processors="fop"
    template="views/blog-post-fo.tpl" />
```

Finally, we create the Smarty XSL-FO template shown in Listing 10. Now, all we need to do in order to render the PDF blog post view is something like this:

```
$dao = SRA_DaoFactory::getDao('BlogPost');
if (BlogPost::isValid($post =
  $dao->findByPk($_GET['postId']))) {
    $post->render('pdf');
}
```

When **$post->render('pdf')** is invoked, sierra-php will first use Smarty to render **views/blog-post-fo.tpl** to a temporary file **$outputFile**, and then run **/usr/local/bin/fop** with two arguments. The first argument is the path to **$outputFile**, and the second is the path to a random temporary file that sierra-php automatically creates: **$randomFile1**. When FOP runs, it will read the XSL-FO document **$outputFile** and use it to create a PDF document **$randomFile1**. Then, sierra-php will output the HTTP header **Content-type: application/pdf**, followed by the contents of the PDF file. Finally, sierra-php will clean up the temporary files.

Adding Controllers

Now that we have defined the model and views for our application, we just need to tie them together by writing a few *controllers*. In the context of this sierra-php, a *controller* is simply a web-accessible PHP script that will act as the glue between a class in the model, one of the views defined for that class, and, optionally, a user action. Listing 11 is the controller for creating a new blog post, and Listing 12 is the wrapper HTML template displayed by that controller (which in turn, renders the **BlogPost** *form* view).

Summary

My intent in writing this article was to demonstrate some of the more useful features provided by sierra-php, as well as the arm-bending required in order to use it. If you would like to learn more about sierra-php, visit the Google Code project website http://sierra-php.org. The complete blog tutorial, demo link and source archive download are provided on the *BlogTutorial* Wiki page. The sierra-php API website (see Related Links) provides full API and DTD documentation including the sierra-php model DTD **entity-model.dtd**.

Happy coding!

JASON READ has worked as web developer since 1998. He holds a Bachelor degree in Computer Science from BYU. Jason currently resides in Provo, Utah, where he works as a freelance developer and technology consultant by day and a human jungle gym for his four small children by night.